2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

A Stable Network-Aware VM Placement for Cloud Systems

Ofer Biran[1], Antonio Corradi[2], Mario Fanelli[2], Luca Foschini[2], Alexander Nus[1], Danny Raz[3], Ezra Silvera[1]
[1] IBM Haifa Research Lab, Haifa, Israel
[2] Dipartimento di Elettronica, Informatica e Sistemistica (DEIS), University of Bologna, Italy
[3] Department of Computer Science, The Technion, Haifa, Israel
[1]{biran, alexnus, ezra}@il.ibm.com, [2]{antonio.corradi, mario.fanelli, luca.foschini}@unibo.it, [3]danny@cs.technion.ac.il

***Abstract*— Virtual Machine (VM) placement has to carefully consider the aggregated resource consumption of co-located VMs in order to obey service level agreements at lower possible cost. In this paper, we focus on satisfying the traffic demands of the VMs in addition to CPU and memory requirements. This is a much more complex problem both due to its quadratic nature (being the communication between a pair of VMs) and since it involves many factors beyond the physical host, like the network topologies and the routing scheme. Moreover, traffic patterns may vary over time and predicting the resulting effect on the actual available bandwidth between hosts within the data center is extremely difficult.**

We address this problem by trying to allocate a placement that not only satisfies the predicted communication demand but is also resilient to demand time-variations. This gives rise to a new optimization problem that we call the Min Cut Ratio-aware VM Placement (MCRVMP). The general MCRVMP problem is NP-Hard; hence, we introduce several heuristics to solve it in reasonable time. We present extensive experimental results, associated with both placement computation and run-time performance under time-varying traffic demands, to show that our heuristics provide good results (compared to the optimal solution) for medium size data centers.

Keywords: Data Center Networking, Cloud Computing, Network-Aware VM Placement, Min Cut Ratio

I. INTRODUCTION

Cloud computing architectures have received an increasing attention in recent years due to their great promises [1, 2]. Cloud providers take advantage of virtualization technologies to gain economical revenues from underutilized IT resources [3]. However, a core management problem at the infrastructure level is the placement of the VMs on physical hosts *(VM placement)*. The goal is to optimally exploit available host resources, while avoiding severe performance degradation due to aggregated resource consumption of co-located VMs.

Several recent works addressed VM placement by considering only local physical node resource constraints, namely CPU and memory sharing, and several algorithms have been proposed with different objectives, such as minimizing the number of powered on hosts [4, 5, 6]. To the best of our knowledge, only few proposals, such as [7] and [8], explicitly considered the effects of network requirements and constraints on VM placement. This aspect, typically referred to as *network-aware VM placement*, is still widely unexplored and there are several open issues, resulting from the following observations. First, from a networking viewpoint, modern data centers use non-trivial topologies that connect physical hosts through multiple paths for the sake of scalability and reliability [9, 10]. Second, they adopt dynamic multi-path routing schemas to exploit the full available bandwidth; hence, traffic demands routed through network paths can dynamically change at run-time. Finally, Cloud data centers host heterogeneous services that result in a large variety of possible run-time traffic patterns. Moreover, this traffic demand suffers from a high variability due to either unpredicted request spikes or service-dependant operations, such as database replication.

This paper focuses on network-aware VM placement in modern Cloud data centers and proposes novel solutions to address all above issues. We introduce a new network-aware VM placement optimality problem, called Min Cut Ratio-aware VM Placement (MCRVMP). MCRVMP considers both constraints on local physical resources, such as CPU and memory, and constraints on network resources evolving from complex network topologies and dynamic routing schemas. This is done by exploiting the notion of network graph cuts to express the latter. Most important, starting from the realistic assumption that traffic demands are time-varying, MCRVMP strives to minimize the maximum ratio of the demand and the capacity across all cuts in the network, in order to find placement solutions that, by having spare capacity on each network cut, can absorb unpredicted traffic bursts.

Since the general MCRVMP problem is NP-Hard even on trees with a single route for each traffic destination (details are omitted due to lack of space), we cannot expect to optimally solve it for large data center. Hence, we propose several heuristics, trading off solution optimality with solving time. Note that our heuristics target realistic data center scenarios and thus we take advantage of typical traffic patterns where communication is done only between VMs belonging to the same service (user). As shown in Section IV, our heuristics can scale up to hundreds of hosts and thousands of VMs, thus making our formulation suitable for medium size data centers. In addition, to better evaluate the validity of our proposal, we exploit NS2 simulations showing that MCRVMP-based placements can efficiently handle time-varying traffic demands with reduced dropped packets.

The paper is organized as follows. In Section II we present

978-0-7695-4691-9/12 $26.00 © 2012 IEEE
DOI 10.1109/CCGrid.2012.119

the most common data center network topologies and define the MCRVMP problem. In Section III we describe our heuristics, and in Section IV we present a comprehensive set of experimental results to assess their technical soundness. Finally, we describe related work and present our conclusions.

II. VIRTUAL MACHINE PLACEMENT

In this section we provide the needed background about our work. First, we introduce several common network topologies for data centers and we show that, for the sake of the MCRVMP problem, it is possible to transform them to equivalent tree topologies. Then, we formally define the MCRVMP problem in the case of tree-based networks.

A. MCRVMP for Data Center Network Topologies

Cloud data centers host heterogeneous services that can lead to very different communication patters, from one-to-one to all-to-all traffic matrixes. In addition, network topology greatly affects the maximum achievable bandwidth with specific traffic patterns. To accommodate that wide range of requirements, a plethora of solutions has been presented in the literature for specific classes of services [9, 10, 11, 12].

Most of the network topologies for data centers share a three-tier architecture [11]. The lowest *access tier* contains physical hosts that connect to access switches. The intermediate *aggregation tier* contains aggregation switches useful to connect together access switches, in order to enable more localized communications among hosts. Finally, the highest *core tier* contains core switches that connect aggregation switches; this tier also includes gateways for the traffic with the outside of the data center. Despite the multitude of proposals, three main topologies are emerging as standard-de-facto solutions: Tree, Fat-Tree, and VL2 (see Fig. 1).

Tree networks are appealing due to their simplicity of wiring and reduced economical costs [11]. Unfortunately, they suffer scalability bottlenecks since the links closer to the tree root have to carry traffic coming from big portions of the data center. We remark that tree-based topologies can be the result of adopted routing protocols; in fact, even if the network topology is a general graph, the usage of VLANs and spanning tree routing algorithms leads to tree logical networks.

Fat-tree is a three-tier topology widely based on bipartite graphs [10]. The basic building block of this topology is the so-called pod (see dotted areas in Fig. 1), defined as a collection of access and aggregation switches connected in a complete bipartite graph. Each pod is connected to all core switches, but links are evenly spread between the aggregation switches contained into the pod; this results in a new bipartite graph between aggregation and core switches. In this topology, all the switches have the same number of ports (e.g., N); for instance, Fig. 1 shows a fat-tree with N equal to 4. The main advantage of this topology is that $N^2/4$ paths are available to route the traffic between two physical hosts.

Finally, VL2 is a three-tier architecture a bit similar to fat-tree [9], but it has a complete bipartite graph between core and aggregation switches, and not between aggregation and access switches. In addition, VL2 introduces a new routing schema, named Valiant Load Balancing, that forces packets received at the access switches to be forwarded up to the core tier. Hence, even if two hosts have a common aggregation switch as parent, packets between them are forwarded first to a randomly selected core switch, and then back to the real destination. That routing schema should grant the best load balancing between all links for unpredictable traffic [9].

Our MCRVMP strives to reduce the worst case cut load ratio, in order to support time-varying traffic demands with reduced packet droppings and no additional VM relocations. However, general graphs can present an exponential number of cuts, but it turns that we do not need to consider all of them from the aspect of network bottleneck. We are only interested in network cuts that partition the set of hosts into two non-empty connected subsets, which are bottlenecks for the traffic demands between VMs placed in different sides of the cut. In the remainder, we call them *critical cuts*. It can be shown that the critical cuts represent all the real bottlenecks of the network, and constraints imposed by other cuts are redundant (e.g., in Fig. 2, the cut $\{l_2, l_3\}$ which partitions the hosts into 3 connected subsets does not contribute a real new constraint above the constraints imposed by the critical cuts $\{l_2\}$, $\{l_3\}$). A critical cut with a load ratio close to 1 means that all the traffic demands carried through it have little degree of variability before leading to dropped packets.

In a tree network, we have a critical cut for each network link. The removal of one link partitions the network and leads to two different subsets of connected hosts. Instead, both fat-tree and VL2 topologies present several bipartite graphs that lead to a higher number of network cuts, thus making the cuts analysis more complex. However, in the case of MCRVMP, it is possible to transform fat-tree and VL2 topologies into *equivalent tree topologies*. In the following, due to space

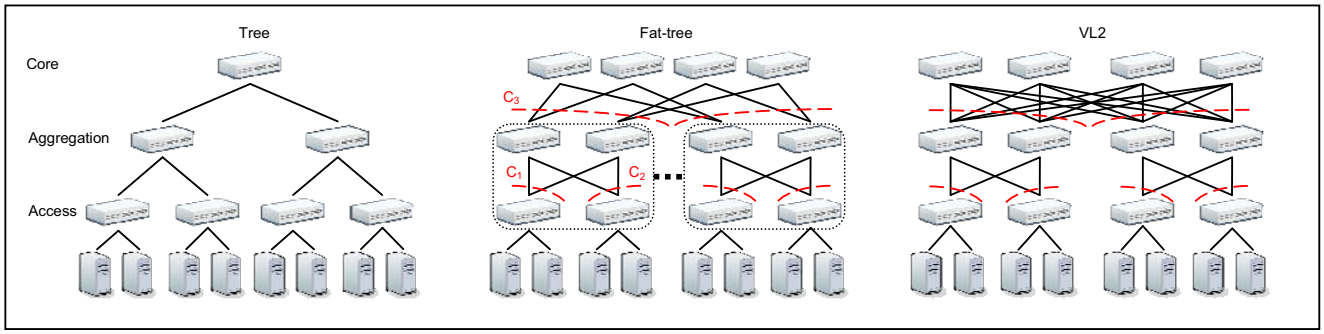

Fig. 1. Common data center network topologies.

constraints, and since algorithms can be easily derived, we only give the main guidelines behind our topology transformations.

Let us focus on the fat-tree topology in Fig. 1. For each pod, under the assumption of dynamic routing, we can study only a limited number of cuts: in particular, for each access switch, we can define a cut (see C_1 and C_2) containing all the links toward the aggregation tier; then, we can define an additional cut (see C_3) by removing all the links going out of the pod. By iterating these rules on all the pods, we can find the critical network cuts for a fat-tree topology. Similarly, for a VL2 topology, we can introduce a cut for each access switch by removing all the links to the aggregation tier. Then, depending on the actual wiring, we can identify bipartite graphs of access and aggregation switches: for each one of them, we introduce an additional cut containing all the links between it and the core tier. Of course, for both fat-tree and VL2 topologies we also consider the network cuts associated with the links between physical hosts and access switches.

Once the critical network cuts are identified, each of them can be replaced, for the sake of MCRVMP, by a single link with equivalent capacity. For the sake of clarity, let us assume that the network topologies in Fig. 1 have all the links with the same capacity L_{CAP}. In this case, the fat-tree can be transformed into an equivalent tree where core and aggregation switches are connected by links with $4*L_{CAP}$ capacity, and aggregation and access switches are connected by links with $2*L_{CAP}$ capacity; similarly, the VL2 topology can be transformed into an equivalent tree where links respectively have $8*L_{CAP}$ and $2*L_{CAP}$ capacities. Next, MCRVMP is solved on the equivalent tree and the obtained placement solution, as well as the load values on the network cuts, applies for the initial topology. Hence, in this paper we consider only tree-based networks, but due to aforementioned transformations obtained results apply also to fat-tree and VL2 topologies.

B. Min Cut Ratio-aware VM Placement (MCRVMP)

MCRVMP considers that traffic demands vary in time because several factors, such as specific time-of-the-day hours and periodic service tasks, can deeply affect the network traffic at run-time. Hence, MCRVMP tries to find placement solutions resilient to traffic bursts in deployed services, by minimizing the maximum load ratio over all the network cuts. More formally, *MCRVMP finds a VM-to-host placement that, while respecting resource constraints on host CPU, host memory, and network cuts, minimizes the maximum network cut load.*

In finer details, MCRVMP addresses the problem of the placement of VMs to hosts belonging to a data center. We consider a set of *n* host $H=\{h_i\}_{i=1,\ \dots n}$ and a set of *m* VMs $=\{vm_j\}_{j=1,\ \dots m}$. Each host h_i is associated with resource capacities $\langle CPU_{CAP}, MEM_{CAP}\rangle$, and each VM vm_j is described by resource requirements $\langle CPU_{REQ}, MEM_{REQ}\rangle$. We use a traffic matrix T to represent average traffic rates between VMs; the element t_{ij} is the traffic rate from vm_i to vm_j. Finally, X is an m*n matrix of binary variables that represent VM placement; the element x_{ij} is 1 if vm_i is placed on host h_j, 0 otherwise.

The data center network topology is a tree (in case of fat-tree and VL2, above graph transformations have to be applied first). Hence, any critical network cut contains only one link, and has a total capacity equal to the capacity of the single contained link. Moreover, each cut partitions the set of hosts H in two disjoint subsets, namely H_1 and H_2; for the sake of clarity, H_1 always contains the hosts of the sub-tree originating from the link endpoint farther from the original tree root. In a valid placement, the aggregated traffic flowing from VMs placed in H_1 to VMs placed in H_2 must be lower than the cut capacity (the same applies for the traffic from H_2 to H_1). Finally, a Cut Load Ratio (CLR) vector contains all the cut load ratio values. For each cut, CLR has two elements: the first one is the ratio between the traffic from H_1 to H_2 divided by the cut capacity; the second one is similar but considers the traffic from H_2 to H_1. CLR size is twice the number of cuts: the entries for H_1 to H_2-traffic are stored in the first half of the vector and all the others in the second part.

To express all critical cuts, we introduce a cut matrix C where each row represents a network cut and each column a host; the element c_{di} is 1 if h_i belongs to H_1 when the d^{th} cut is considered. C matrix implicitly represents the data center topology, and has to be generated off-line by computing all the critical cuts. This step is very simple for a tree topology: it only requires to remove one link each time and to express the C row associated with the two host partitions H_1 and H_2. For instance, Fig. 2 shows the cut matrix C associated with a 3-level binary tree; the index of the row is the same one of the link removed to generate the associated cut. By using the cut matrix C, the total traffic rate from H_1 to H_2 (respectively, H_2 to H_1) over the d^{th} cut is the d^{th}-element of the diagonal of the matrix $[C * X^T * T * X * (1-C)^T]$ (respectively, $[(1-C) * X^T * T * X * C^T]$). This accounts only for the traffic among VMs; the traffic from/to the gateway, situated at the tree root, is also added depending on the VMs placed in H_1 (not shown in the following model for the sake of readability).

Hence, we formally express our MCRVMP problem by using the following integer quadratic programming model:

$$\min\left(\max_{c\in 1\dots 2*N_{cut}} CLR_c\right) \tag{1}$$

$$\sum_i vm_i.CPU_{REQ} * x_{ij} \le h_j.CPU_{CAP} \quad \forall j \tag{2}$$

$$\sum_i vm_i.MEM_{REQ} * x_{ij} \le h_j.MEM_{CAP} \quad \forall j \tag{3}$$

$$CLR_c = \begin{cases} \dfrac{[C * X^T * T * X * (1-C)^T]_{cc}}{CAP_c}, \forall c \in \{1,\dots,N_{CUT}\} & (4) \\ \dfrac{[(1-C) * X^T * T * X * C^T]_{cc}}{CAP_c}, \\ \qquad \forall c \in \{N_{CUT}+1,\dots,2*N_{CUT}\} & (5) \end{cases}$$

$$CLR_c \le 1 \ \forall c \in \{1,\dots,2*N_{CUT}\} \tag{6}$$

$$\sum_j x_{ij} = 1 \quad \forall i \tag{7}$$

$$x_{ij} \in \{0,1\} \quad \forall i, \forall j \tag{8}$$

Formulas (2) and (3) enforce CPU and memory capacities at each single host. Formulas (4) and (5) express the aggregate traffic flowing on each network cut, while formulas (6) enforce feasible solutions from the network point-of-view. Formulas (7) avoid the same VM to be placed on multiple hosts, while formula (1) expresses MCRVMP goal, namely to minimize the worst case cut load ratio.

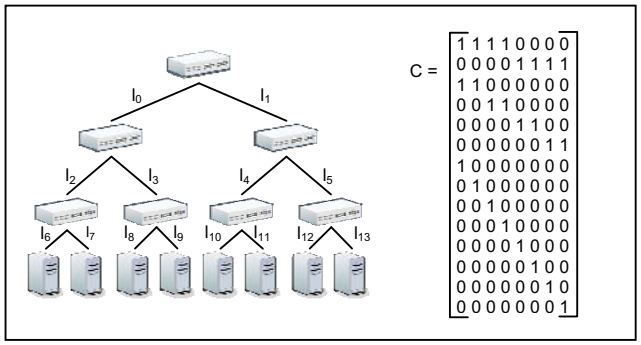

Fig. 2. Cut matrix C for a binary tree topology.

Finally, we clearly remark that the MCRVMP problem formulation presented in this paper is not meant to be applied on-line, namely to serve new VM requests in a data center that already contains placed VMs. In fact, here we focus on finding the optimal MCRVMP solution, while completely leaving out VM relocation costs; those have to be definitely considered in on-line formulations of our optimality problem, so as to prevent frequent and expensive VM migration operations.

III. PLACEMENT ALGORITHMS

MCRVMP is an NP-hard problem that can be optimally solved only for small problem instances. Hence, to make it suitable for real-world Cloud scenarios, we designed two heuristics, called *2-Phase Connected Component-based Recursive Split (2PCCRS)* and *Greedy Heuristic (GH),* with diverse tradeoffs between solution quality and execution time.

Both our heuristics take advantage of the fact that Cloud scenarios feature Connected Components (CCs) of VMs that exchange data only between themselves or with the external gateway, such as the case where multiple VMs of the same customer run a three-tier web application. Hence, it is reasonable to cluster VMs in CCs so that, by considering a CC as a single VM to be placed, we can reduce problem complexity. In addition, since tree networks suggest the use of recursive algorithms where intermediate placement sub-problems partially fix VM-to-host assignments, 2PCCRS uses this property to reduce the solution space; this leads to a reduced applicability since 2PCCRS can be applied only if the network topology is (or can be transformed into) a tree.

A. 2PCCRS Placement Algorithm

Our first placement algorithm, called 2PCCRS, exploits integer programming techniques to solve the MCRVMP problem. The adopted name recalls its two main properties. First, it is recursive: it exploits the tree network structure to define and solve small problem instances on one-level trees. Second, it adopts a two-phase approach: the first phase places CCs in the network, while the second phase expands them to place actual VMs on physical hosts. Both these properties allow 2PCCRS to deal with MCRVMP complexity: each placement step, solved by a mixed integer programming solver, has to deal with a reduced number of VMs, hosts, and network topology cuts.

In a more detailed view, the first phase places CCs on network switches, meaning that VMs belonging to particular CCs have to be mapped in the sub-tree rooted at the associated network switch. By processing the matrix T, 2PCCRS clusters VMs in $CC=\{cc_d\}_{d=1,\dots,t}$. Each cc_d is associated with a vector of resource requirements $\langle CPU_{TOT}, MEM_{TOT}, IN_{TOT}, OUT_{TOT}\rangle$, representing aggregated requirements in terms of CPU, memory, download and upload traffic with the gateway. Since CCs do not exchange traffic among them, at this phase we need to only model the traffic with the gateway. According to the recursive approach, 2PCCRS starts from the tree root, and solves an initial placement step by considering the one-level tree associated with the tree root. It considers a set of virtual hosts $VH=\{vh_z\}_{z=1,\dots,q}$, one for each aggregation switch, that represents the total capacity available in the sub-tree rooted at the switch; hence, vh_z capacities are equal to the aggregated capacities of all the hosts in the sub-tree. As output of this placement step, we have cc_d-to-vh_z associations, meaning that VMs belonging to cc_d will have to stay in the sub-tree rooted at vh_z. It could be that few cc_d are not placed due to resource capacities; hence, we consider them placed on the real tree root, meaning that their own VMs can be placed on all the hosts in the second phase. Then, 2PCCRS solves placement sub-problems associated with the one-level trees where aggregation switches are roots (similarly virtual hosts are defined for the associated sub-trees). At the end of the first phase, 2PCCRS has cc_d-to-vh_z associations, where vh_z can be associated to the core switch, one of the aggregation switches, or one of the access switches; each association means that, in the second phase, cc_d's VMs have to be placed in the sub-tree rooted at the associated network element.

At each placement step, 2PCCRS invokes a mathematical solver to obtain the current placement solution. However, the placement problem solved by 2PCCRS during the first phase differs from MCRVMP. In fact, due to resource constraints, it could be impossible to place all the involved CCs at each step: hence, $\sum_z x_{dz} = 1\ \forall d$ needs to be relaxed into $\sum_z x_{dz} \leq 1\ \forall d$ to obtain feasible results. However, that leads to useless results since the solver will reach the solution with all the CCs not placed and minimum worst case cut load ratio of 0. Hence, we associate each cc_d with a penalty traffic that is added on all the network cuts when cc_d is not placed. In a partial solution, if cc_d is not placed, it will be split in different sub-trees, meaning that part of its own internal traffic will load the cuts at the current level. To favor the placement of CCs likely to lead to high cut load values, cc_d penalty traffic is the sum of the average of internal traffic demands and traffic with the gateway. In this way, if possible, the solver will always place a cc_d to reduce the worst case cut load ratio. More formally, the placement sub-problem solved at each step is represented by the following integer linear mathematical model (formula (9)-(18)):

$$\min\left(\max_{c\,\in 1\dots 2*N_{cut}} CLR_c\right) \tag{9}$$

$$\sum_d cc_d.CPU_{TOT} * x_{dz} \leq vh_z.CPU_{CAP}\ \forall z \tag{10}$$

$$\sum_d cc_d.MEM_{TOT} * x_{dz} \leq vh_z.MEM_{CAP}\ \forall z \tag{11}$$

$$isCCPlaced_d = \sum_{\forall z} x_{dz} \quad \forall d \tag{12}$$

$$penaltyT = \sum_{\forall d}(1 - isCCPlaced_d) * cc_d.penaltyTraffic \tag{13}$$

$$\mathrm{isBelowCut}_{dc} = \sum_{\forall z \text{ in } H_1(c)} x_{dz} \quad \forall d, \forall c \tag{14}$$

$$\mathrm{CLR}_c = \begin{cases} \dfrac{\sum_{\forall d}[cc_d.\,\mathrm{IN}_{TOT} * (1 - \mathrm{isCCPlaced}_d + \mathrm{isBelowCut}_{dc})] + \mathrm{penaltyT}}{\mathrm{CAP}_c} \\ \forall c \in \{1, \dots, N_{CUT}\} \\ \dfrac{\sum_{\forall d}[cc_d.\,\mathrm{OUT}_{TOT} * (1 - \mathrm{isCCPlaced}_d + \mathrm{isBelowCut}_{dc})] + \mathrm{penaltyT}}{\mathrm{CAP}_c} \\ \forall c \in \{N_{CUT} + 1, \dots, 2 * N_{CUT}\} \end{cases} \tag{15}$$

$$\mathrm{CLR}_c \leq 1 \; \forall c \in \{1, \dots, 2 * N_{CUT}\} \tag{16}$$

$$\sum_z x_{dz} \leq 1 \quad \forall d \tag{17}$$

$$x_{dz} \in \{0, 1\} \quad \forall d, \forall z \tag{18}$$

In the second phase, 2PCCRS splits the CCs to place the real VMs. This phase adopts a recursive approach similarly to the previous phase, but it solves real MCRVMP problems as defined in Section II. At the beginning, when it solves the tree root, 2PCCRS considers all the VMs of CCs that, in the first phase, have been forced to stay there due to resource constraints. As in the previous phase, each aggregation switch is associated with a VH that represents the total capacity in the sub-tree. However, we subtract the aggregated CPU and memory requirements of the CCs placed into the sub-tree, and we add their traffic demands with the gateway. Output of this initial step is a placement of these VMs on the VHs representing the aggregation switches. Then, 2PCCRS solves the placement sub-problems where the aggregation switches are roots. Each of these sub-problems handles the VMs that were placed on the VH associated with the current root switch, and VMs belonging to CCs placed on the VH associated with the current root switch during the first phase. Similarly to the previous phase, 2PCCRS keeps solving intermediate placement sub-problems up to the leaves, where we finally have VM-to-host associations.

Regarding 2PCCRS complexity, there are few important things to highlight. First, since CCs do not have traffic demands between themselves, the placement sub-problems of the first phase are integer linear (not quadratic) programming problems; also, the number of CCs is much smaller than the number of VMs. Second, thanks to the first phase, the second 2PCCRS phase usually has to solve small problem instances. For instance, if we consider the placement sub-problem associated with the tree root, 2PCCRS does not initially consider all the VMs of CCs that were placed in VHs in the first phase.

B. GH Placement Algorithm

We also present a second placement algorithm, called Greedy Heuristic (GH), which completely leaves out mathematical programming and greedily places VMs on available hosts. Differently from 2PCCRS, where intermediate sub-problems fix VMs to be placed in sub-trees, this heuristic places each VM individually, thus having more freedom during placement. In brief, GH consists of two main phases: the first one ranks all the traffic demands, while the second one exploits the ranking to place VMs on available hosts.

We first provide some useful notations. As VMs are iteratively placed, it is possible that a placed VM has traffic demands from/to VMs not placed yet: that traffic demand is called *floating* when we cannot determine a-priori the affected network cuts. On the other hand, if a traffic demand has both end-points placed, we call it *committed* since the affected network cuts are determined. Finally, during placement computation, *a traffic demand is committed by a VM-to-host placement* if its status changes from floating to committed due to the current placement operation.

In the first phase, GH ranks all the traffic demands. It starts by extracting the CCs out of the traffic matrix T. Then, it ranks the CCs to find the ones that will be worth not to split from the point of view of MCRVMP objective function. Toward this goal, it orders all the traffic demands by decreasing values, and associates each cc_d with an accumulator whose value is the sum of the relative positions occupied by the demands within cc_d in the ranked list. Intuitively, the higher the accumulator value, the higher the number of big flows within cc_d, hence, its splitting will have a bigger effect on cuts load. Finally, GH orders CCs by decreasing accumulator values, and then, following this order, extracts the traffic demands; for each cc_d, demands are considered in decreasing order.

In the second phase, GH iteratively processes the ranked traffic demands. For each traffic demand, it has to initially select the VM to place. If both the VMs involved in the traffic demand have been already placed, it skips to the next demand; if only one of them has been placed, it considers the remaining one; finally, if both VMs are not placed yet, it considers the one that, after the current placement operation, would commit the higher number of demands. Then, GH filters all the hosts to consider only the ones having enough resource capacities to accommodate the current VM. It iteratively tries to place the VM on each feasible host, while evaluating all the network cut values. At the end, the VM is placed on the host that will lead to the minimum value of the maximum cut load value. GH iterates above steps until all the VMs are placed.

During the ongoing placement process, we also consider for each cut the future effect of the floating traffic demands, as part of them might be routed over the cut. This is more likely to happen for floating traffic demands with the already-placed VM in a partition with low residual capacities, as the pair VM will be probably placed in the opposite partition. For this, we estimate the final traffic over each cut as a weighted sum of committed and floating demands. The weight of a floating demand (between 0 and 1) is anti-proportional to the worst case ratio (i.e., closer to 0) between residual and total capacities of both CPU and memory resources in the partition where the already-placed VM is.

In the next section, we compare 2PCCRS and GH to highlight principal tradeoffs between their solution quality and execution time.

IV. Experimental Results

In order to test the expected performance of our placement algorithms, and to validate that our approach can indeed improve resilience to time-varying traffic demands, we present two sets of experimental results. First, in Section IV.A, we focus on MCRVMP-based placement computation by comparing the random, optimal, 2PCCRS, and GH solutions.

Then, in Section IV.B, we show through NS2-based simulations that MCRVMP-based placement solutions can effectively absorb time-varying traffic demands, and are indeed resilient to traffic changes, thus increasing placement stability.

A. Comparisons between Placement Heuristics

Here, we compare our two heuristics, 2PCCRS and GH, by focusing on placement quality and solving time. To better assess our proposals, we also consider two additional algorithms. The first one, called Random (RND), randomly generates VM-to-host assignments; it is interesting to compare MCRVMP-based placements with a network-oblivious one. The second one, called Optimal (OPT), uses a mixed integer programming solver to solve the entire MCRVMP problem, as detailed in Section II; hence, it finds the optimal solution, i.e., the VM placement that minimizes the worst case cut load ratio. Due to the associated complexity, experimental results for the OPT algorithm are available only for extremely small and unrealistic problem instances.

All the following experimental results are associated with a data center made by a pool of homogeneous hosts having the same capacity for CPU and memory resources. In addition, all VMs have equal CPU and memory requirements; hence, due to capacity constraints, each host in the pool can accommodate the same number of VMs. The data center network is always a fully balanced tree with link capacity of 1 Gbps. We execute our heuristics on a physical server with CPU Intel Core 2 Duo E7600 @ 3.06GHz and 4 GB RAM, and we exploit IBM ILOG CPLEX as mixed integer mathematical solver to compute OPT solutions and solve the intermediate steps of 2PCCRS. ILOG is always configured with pre-solve and parallel mode enabled; due to hardware limitations, it exploits a maximum of 2 threads during solving. Finally, all the reported experimental results are average values over 10 different executions; in addition, we report standard deviation values to better assess the confidence of our results.

One crucial aspect is the modeling of the traffic matrix T. We have to produce CCs but, at the same time, we need to test our heuristics with different T as the total number of considered traffic demands greatly affects problem complexity. Hence, we generate T taking into account three main parameters: 1) CCs size; 2) traffic patterns between VMs of the same CC; and 3) rate of the traffic demand (in Mbps). Due to space constraints, we focused our evaluation on one challenging and realistic case study. For CCs size, we consider them distributed according to a uniform distribution. Then, traffic demands between VMs in the same CC are randomly generated with a probability lower than 1, and with rate following a Gaussian distribution (mean = 5 Mbps, standard deviation = 0.5 Mbps). Also, each CC has a VM with both upload and download traffic demands to the gateway, with rates generated according to another Gaussian distribution (mean = 2 Mbps, standard deviation = 0.2 Mbps).

The first set of experiments focuses on small problem instances in order to be able to compare our heuristics with the OPT algorithm. The experimented network topology is a three-level binary tree; 24 VMs have to be placed on 8 hosts under different traffic matrices. Here, we consider CCs size uniformly distributed in [1; 8]; inter-VMs traffic demands are randomly generated with a probability in {0.5, 0.75}. Fig. 3.a, 3.b, and 3.c respectively show the worst case cut load value, the average cut load, and the placement computation time. Fig. 3.a shows that both 2PCRRS and GH reach worst case cut load values very close to the OPT algorithm, while RND is the worst one. We remark that worst case cut load ratios are very small due to the reduced number of considered traffic demands. In Fig. 3.b, we note that, to minimize the worst case cut load ratio, OPT leads to average link load values slightly higher than the ones produced by our heuristics. Hence, our heuristics carry less traffic into the data center than OPT (not shown in figure), but they have higher worst case cut load ratios. Finally, we evaluated placement computation time: RND, 2PCCRS and GH have placement computation times close to zero for these little problem instances; instead, OPT, as it can be seen in Fig. 3.c, presents extremely high computation times, that make OPT completely unfeasible for real Cloud scenarios. In addition, OPT placement computation times have an extremely high standard deviation, meaning that the number of actual traffic demands greatly affects the capability of the solver to quickly find the optimal. In brief, OPT computation time is not only very long, but also difficult to predict.

The second set of experiments considers a wider network deployment by using a fully balanced quaternary tree with 64 hosts. In this case, we increment the number of VMs (from 2x to 20x the number of hosts), to compare heuristic scalability; as regards traffic matrix, CCs size follow a uniform distribution in [1; 16], while associated traffic pairs are generated with a probability of 0.75. Fig 4.a, 4.b, and 4.c respectively show the same set of results used in the previous case for this new scenario. RND algorithm is not showed here since it was able to reach feasible placements (with worst case cut load ratios higher than 0.9) only for the case of 128 VMs; apart from that, it always reached unfeasible placements with cut load values higher than 1. In Fig. 4.a and 4.b, we remark that 2PCCRS and GH reach similar results for small numbers of VMs; then, starting from 640 VMs, 2PCCRS always performs significantly better than GH. In Fig. 4.b, we note that 2PCCRS also reaches lower average link loads. Although 2PCCRS leads to better VM placement solutions, it has extremely higher computation

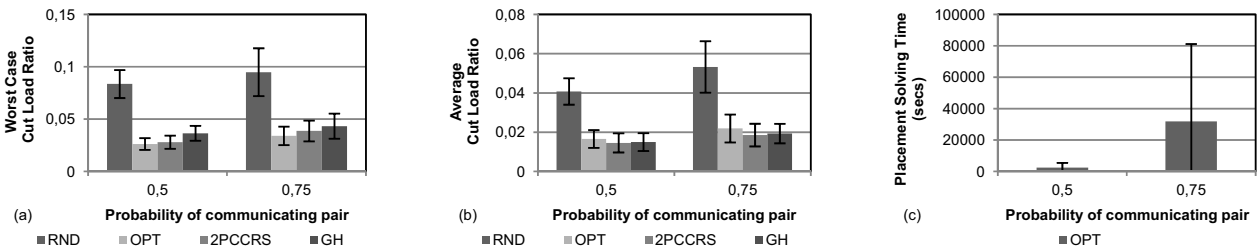

Fig. 3. Worst case cut load value (a), average cut load values (b), and placement computation times (c) for a small topology of 8 hosts.

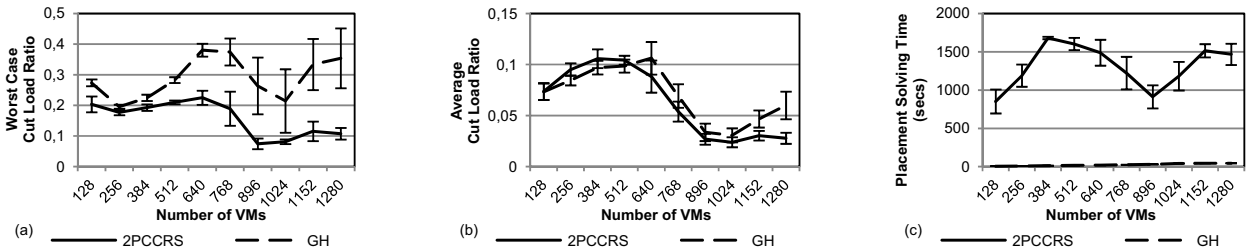

Fig. 4. Worst case cut load value (a), average cut load values (b), and placement computation times (c) for a topology of 64 hosts.

times than GH. While GH placement times increase almost linearly with the number of VMs (in the worst case scenario, it computes the placement in about 50 secs), 2PCCRS computation time is higher due to the usage of mathematical programming techniques (see Fig. 4.c). With aforementioned numbers of VMs, solving time increases remarkably, since each 2PCCRS placement step actually tries to find the optimal solution; at the same time, the solver typically finds very good results in the very first optimization steps, and then it only obtains limited improvements when run for longer time spans. In our experiments, we limit maximum placement computation time to 1800 seconds because we found that this total solving time ensures a good tradeoff between solution quality and placement computation time. Similarly to the previous scenario, the execution times of the solver are not predictable and depend on the specific problem instance; the case with 384 VMs was actually the one with longest solving time.

Finally, the last set of experiments evaluates heuristics scalability as the data center grows. We fixed a number of VMs per host equal to 10, and we scaled the data center topology from 64 to 343 hosts by always having fully balanced trees; hence, we experimented with 640 to 3430 VMs. For the traffic matrices, we used the same parameters of the previous experiments. Fig. 5.a shows the worst case cut load ratios achieved by our heuristics: we can see that 2PCCRS usually performs better than GH. In addition, Fig. 5.b shows the total placement computation times of the two heuristics. While GH is faster than 2PCCRS for small data center sizes, it is much

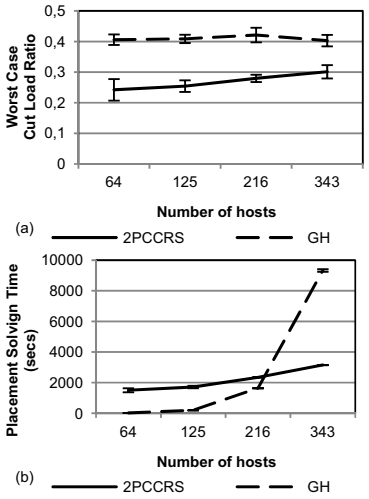

Fig. 5. Worst case cut load value (a), and placement computation times (b) for large-scale topologies.

more sensitive to topology scaling. This is mainly due to the fact that GH considers all the hosts and all the network cuts. For instance, in the case of 343 hosts, GH considers all the 343 hosts and 56 network cuts each time it needs to place a VM; instead, by focusing on one-level trees, in the worst case 2PCCRS considers only 7 virtual hosts and 7 network cuts. Finally, we note that, even if we limit the solving time to 1800 seconds, 2PCCRS reaches total placement computation time of about 3000 seconds (see Fig. 5.b). This is consequence of the additional time spent by CPLEX solver to pre-process the mathematical problem before starting the real optimization.

We conclude that both 2PCCRS and GH can reasonably solve MCRVMP with different tradeoffs between solution quality and placement computation time. 2PCCRS always reaches lower worst case cut load ratios, and scales better with topology size, while GH is significantly faster for small data center topologies.

B. Placement Validation with NS2 Simulations

We used NS2 to better assess the resilience of MCRVMP-based placement solutions under time-varying traffic demands. Due to space constraints, we focus on the case of 64 hosts and 128 VMs (see Fig. 4). We selected that specific case since 2PCCRS and GH have different worst case cut load ratios (see Fig. 4.a), but similar average cut load ratios (see Fig. 4.b); in this way, we aim to find performance indicators that mainly depend on the worst case cut load ratio. Then, for each placement solution, we remove traffic demands between VMs co-located on the same host; each remaining demand is mapped in NS2 through an UDP source/sink pair. For each placement solution, we run 10 simulations with different seeds, thus having a total of 100 runs for each case study; in the remainder, we show average values and standard deviations of all the considered simulations. Finally, each NS2 simulation lasts 3600 seconds.

Each source produces a constant traffic rate according to the demand contained in the traffic matrix, by emitting UDP packets of 60KB each. Then, after D_{LOW} seconds, the source increases the traffic rate to R times the nominal value, and this increased demand lasts for D_{HIGH} seconds. This process repeats for the whole simulation, thus having normal and high traffic rates interleaved by D_{LOW} and D_{HIGH} times. D_{LOW} (respectively, D_{HIGH}) values are produced by a Gaussian distribution with mean of 200 seconds and standard deviation of 20 seconds (respectively, 100 seconds and 10 seconds for D_{HIGH}).

Fig. 6.a and 6.b respectively show the percentage of dropped packets and the average packet delivery delay for R in {1, 3, 5, 7}. Both 2PCCRS- and GH-based placements can

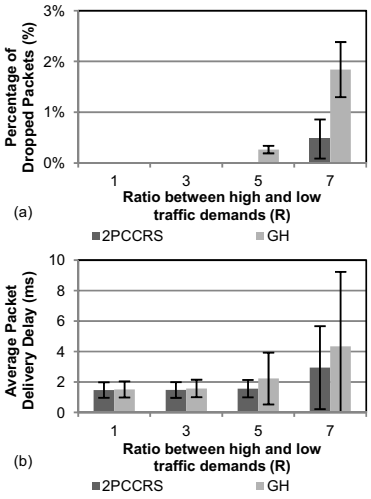

Fig. 6. Percentage of dropped packets (a), and average packet delivery delay (b) in NS2 simulations.

absorb traffic demands up to three times the nominal values with no dropped packets. When R is 5, GH-based placements start experiencing dropped packets. In fact, from Fig. 4.a, we note that such solutions have a worst case cut load ratio close to 0.3; hence, when R is 5, the worst case cut (and the ones with similar load values) will be likely to be congested. Similarly, 2PCCRS-based placements experience dropped packets when R is 7. Finally, Fig. 6.b shows that 2PCCRS-based placements have average packet delivery delays lower than GH-based ones, due to the less loaded network cuts.

To conclude, statistically speaking and considering that average cut load ratios are similar between 2PCCRS- and GH-based VM placements, performance improvements of 2PCCRS over GH are mainly consequence of the reduced worst case cut load ratio; hence, MCRVMP-based placements increase the capability of absorbing time-varying traffic demands.

V. Related Work

In recent years, a lot of work has been done in the area of VM placement for Cloud systems. Different works addressed such topic by considering CPU and memory requirements, while leaving out networking aspects [4, 5, 6]; for the sake of brevity, here we omit them and we focus on a small subset of important related works that, similarly to ours, concentrate on Cloud networking.

In [7], the authors focus on the problem of network-aware VM placement with the goal of reducing the aggregate traffic into the data center. They introduce a new placement problem, called Traffic-aware VM Placement Problem (TVMPP), that belongs to the family of the Quadratic Assignment Problems (QAP). Since TVMPP strives to reduce the aggregate traffic, the adopted cost matrix contains the number of hops between each pair of hosts. As demonstrated by authors, TVMPP is NP-hard, hence the authors introduce a heuristic approach to solve it in a reasonable time. As opposed to MCRVMP, TVMPP exploits a different cost function and places only one VM on each host; in addition, since it does not consider link capacity constraints, it can lead to placement solutions with congested links into the data center.

While both TVMPP and MCRVMP assume static and well-defined traffic demands, in [8] authors focus on the equivalent capacity notion to consolidate VMs with uncorrelated traffic demands. Traffic demands are modeled as stochastic variables, and the optimization problem strives to place VMs while ensuring that the maximum network capacity is not violated with a particular user-defined probability. Hence, the final VM placement problem is a stochastic bin packing problem, and authors introduce a new heuristic approach to solve it. Differently from our MCRVMP formulation, in [8] the authors do not take data center topology into account, and consider traffic demands only to perform host-level consolidation; hence, network capacity violations into the data center are possible.

In [13], the authors consider the placement of applications made by a computation and a storage part. They detail the Coupled Placement Problem (CPP) to place such applications, while reducing the total traffic into the network. CPP exploits a cost function similar to the one adopted by TVMPP, but it considers traffic demands only between the computation and the storage part of the same application; of course, this makes the problem much simpler. Comparing CPP with MCRVMP, they share the quadratic nature of the optimization problem, and they both consider multiple applications-per-host and CPU/memory constraints. However, as mentioned above, CPP is simpler due to the restricted set of considered traffic demands. Also, since the authors do not consider constraints on network capacities in problem formulation, CPP solutions can be unfeasible from the network perspective.

With a similar perspective, in [14], authors consider Cloud data centers with server and storage virtualization facilities, and strive to increase load balancing at multiple layers, including servers, switches, and storage. They propose a new algorithm, called VectorDot, to consider multiple resource dimensions from these layers. Since VectorDot needs to solve multi-dimensional knapsack problem, it is largely inspired by the well-known Toyoda method [15]. Differently from [14], our MCRVMP is explicitly focused on networking aspects, and it leaves out CPU-/memory-related optimization goals. Also, our optimization goal does not ensure load balancing among all the network links.

In [16], authors consider networking aspects but with the main goal of reducing the total energy consumption associated with powered on switches and patches. They present a new Network-Aware Virtual Machine Placement (NAVP) problem and a heuristic approach to solve it. As opposed to our work, NAVP strives to consolidate as much traffic demands as possible over the same set of network links, so as to reduce the total energy consumption. Instead, both MCRVMP and the other related works presented above push toward less energy-friendly solutions as they actually strive to reach some load balancing between all the network paths; those are clearly conflicting optimization goals.

Finally, it is important to remark that similar works have appeared in the close research area of virtual network embedding on shared physical substrate [17, 18, 19]. However, to the best of our knowledge, such solutions map only one

virtual node for each physical one, hence they do not well fit the general Cloud scenario where multiple VMs have to be placed on the same physical host. In addition, these works usually adopt cost functions dealing with economical revenues, and introduce resource reservation mechanisms to enforce granted network bandwidth. Differently, MCRVMP allows multiple VMs-per-host and, since it does not assume static resource reservation mechanisms at the network level, strives to improve VM placement stability and dropped packets by minimizing the worst case cut load ratio.

VI. Conclusion

Smart VM placement algorithms are essential in order to efficiently exploit data centers' computing resources. In this paper, we study stable traffic aware VM placement using the newly defined MCRVMP optimization problem. We presented two novel heuristics and showed that they can be efficiently computed (much better than known techniques for the general Quadratic Assignment Problem) when applied to medium data centers where the traffic is often done only between VMs belonging to the same service. Our experimental results indicate that MCRVMP-based placements increase data center scalability, while being able to support time-varying traffic demands with a reduced number of dropped packets.

Several research directions are left for future work. First, since traffic demands can be correlated, we want to extend MCRVMP to further reduce the amount of dropped packets when multiple VMs present high and correlated traffic. Second, since traffic demands between VMs on the same physical host introduce host CPU and memory overhead, we want to extend MCRVMP to prevent solutions with very high local overhead due to communications.

References

[1] M. Armbrust, A. Fox, R. Griffith et al., "Above the Clouds: A Berkeley View of Cloud Computing," University of California at Berkley, Tech. Rep. UCB/EECS-2009-28, 2009.

[2] R. Buyya, C. S. Yeo, S. Venugopal et al., "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," Elsevier Future Generation Computer Systems, vol. 25, no. 6, pp. 599-616, 2009.

[3] A. Lenk, M. Klems, J. Nimis et al., "What's inside the Cloud? An architectural map of the Cloud landscape." in the Proceedings of the 2009 ICSE Workshop on Software Engineering Challenges of Cloud Computing (CLOUD'09), pp. 23-31, 2009.

[4] M. Sindelar, R. K. Sitaraman, and P. Shenoy, "Sharing-aware algorithms for virtual machine colocation," in Proceedings of the 23rd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'11), pp. 367-378, 2011.

[5] S. Lee, R. Panigrahy, V. Prabhakaran et al., "Validating Heuristics for Virtual Machines Consolidation," Microsoft Research, Tech. Rep. MSR-TR-2011-9, 2011.

[6] C. Isci, J. E. Hanson, I. Whalley et al., "Runtime Demand Estimation for effective dynamic resource management," in Proceedings of the IEEE Network Operations and Management Symposium (NOMS'10), pp. 381-388, 2010.

[7] X. Meng, V. Pappas, and L. Zhang, "Improving the scalability of data center networks with traffic-aware virtual machine placement," in Proceedings of the 29th IEEE Conference on Information Communications (INFOCOM'10), pp. 1154-1162, 2010.

[8] M. Wang, X. Meng, and L. Zhang, "Consolidating Virtual Machines with Dynamic Bandwidth Demand in Data Centers," in Proceedings of the 30th IEEE Conference on Information Communications (INFOCOM'11), pp. 71-75, 2011.

[9] A. Greenberg, J. R. Hamilton, N. Jain et al., "VL2: a scalable and flexible data center network," in Proceedings of ACM SIGCOMM 2009 conference on Data communication (SIGCOMM'09), pp. 51-62, 2009.

[10] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, commodity data center network architecture," in Proceedings of ACM SIGCOMM 2008 conference on Data communication (SIGCOMM'08), pp. 63-74, 2008.

[11] Cisco data center infrastructure 2.5.

[12] C. Guo, G. Lu, D. Li et al., "BCube: a high performance, server-centric network architecture for modular data centers," ACM SIGCOMM Comput. Commun. Rev., vol. 39, no. 4, pp. 63-74, 2009.

[13] M. Korupolu, A. Singh, and B. Bamba, "Coupled Placement in Modern Data Centers," in Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'09), pp. 1-12, 2009.

[14] A.Singh, M. Korupolu, and D. Mohapatra, "Server-storage virtualization: integration and load balancing in data centers." in Proceedings of the 2008 ACM/IEEE conference on Supercomputing (SC'08), pp. 1-12, 2008.

[15] Y. Toyoda, "A simplified algorithm for obtaining approximate solutions to zero-one programming problems," Management Science, vol. 21, no. 12, pp. 1417-1427, 1975.

[16] V. Mann, A. Kumar, P. Dutta, and S. Kalyanaraman, "VMFlow: leveraging VM mobility to reduce network power costs in data centers," in Proceedings of the 10th international IFIP TC 6 conference on Networking - Volume Part I (NETWORKING'11), pp. 198-211, 2011.

[17] N.M.M.K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual Network Embedding with Coordinated Node and Link Mapping," in Proceedings of the 28th IEEE Conference on Information Communications (INFOCOM'09), pp.783-791, 2009.

[18] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in 1st ACM workshop on Virtualized Infrastructure Systems and Architectures (VISA'09), pp. 81-88, 2009.

[19] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," Washington University, Tech. Rep. WUCSE-2006-35, 2006.